# Lecture 6.5 : Gradient Descent의 상세

## Fastcampus Math Camp

신승우

Sunday 1st July, 2018

# Gradient Descent : Linear Regression

우리가 지난번에 다루었던 Linear Approximation의 경우, 다음을 만족하는 $\vec{x}$를 해석적으로 구할 수 있었다.

$$min_{\vec{x}}|A\vec{x} - \vec{b}| \tag{1}$$

이를 만약 Gradient Descent 알고리즘을 이용하여 구한다면, 대략 다음과 같은 과정을 따를 것이다.

- Loss Function $J(\theta)$를 구성한다. 여기서는 $|A\vec{x} - \vec{b}|$가 Loss Function이 될 것이며, $\theta$는 $\vec{x}$로, loss function의 값을 계산하는 파라미터일 것이다.
- $\nabla_\theta J(\theta)$를 계산한다.
- $\theta$의 초기값을 지정한다.
- 데이터를 이용해서 parameter를 조정한다.

# Gradient Descent Variants

이 때, 어떤 식으로 데이터를 이용하고 그래디언트를 구하는지에 따라서 아래의 3가지 변형이 있다.

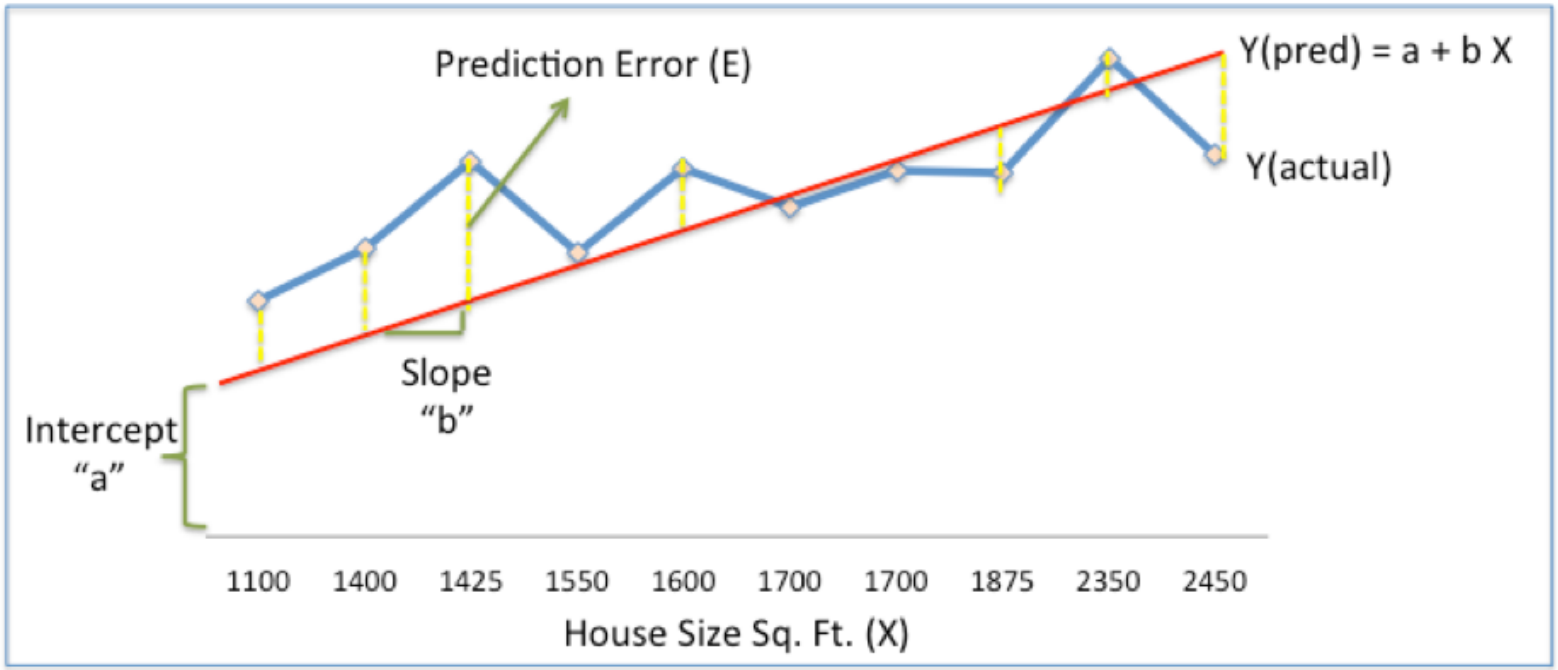
- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent

# Batch Gradient Descent

가장 기본적인 Gradient Descent로, 다음의 과정에 따라 최적값을 찾는다.

- 처음 parameter와 learning rate $\eta$를 정한다.
- 모든 데이터에 대해서 그 지점에서의 gradient descent 값을 결정한다.
- parameter을 $\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta)$와 같이 업데이트한다.

여기서 모든 데이터를 이용한다는 것에 대해서 일견 이해하기 어려울 수 있기 때문에, 선형 회귀의 예시를 들어 설명하고자 한다.




선형 회귀에서의 parameter은 a,b 두 개이다. 이 때, 우리는 loss function을

$$\sum_i ((ax_i + b) - y_i)^2 \tag{2}$$

로 정의하여, 이의 gradient를 구할 수 있다. 이 때 gradient를 구하는 과정은 단순 미분이므로 데이터를 사용하지 않는다. 이후, 저 loss function을 계산할 때 모든 데이터를 다 반영하여 계산하는 것이다.

# Stochastic Gradient Descent

Stochastic Gradient Descent는 이와는 조금 다르다. 이 때는 각 데이터 하나에 대해서만 loss function을 계산한다.

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta; (x_i, y_i)) \tag{3}$$

즉, 위 loss function과는 다르게 $\sum$이 없다. 일반적으로 stochastic gradeint descent는 탐색 과정에서 parameter 값이 크게 출렁이며, 그렇기에 학습이 진행될 때마다 learning rate를 줄여주는 추가적인 optimization을 시행하기도 한다.

# Mini-batch Gradient Descent

Mini-bacth는 두 방법의 혼합으로, 한번에 batch size만큼의 데이터를 이용하여 optimization을 하는 것이다. 즉, loss function을 계산할 때 다음과 같이 하게 된다.

$$J(a, b) = sum_{i=k}^{k+b}((ax_i + b) - y_i)^2 \tag{4}$$

이 때 b는 batch size이다.

# Challenges in Gradient Descent

위와 같은 기본적인 Mini-batch Gradient Descent 알고리즘은 실험적으로 볼 때 항상 잘 converge 하지는 않으며, 이는 아래의 이유들에 기인한다.

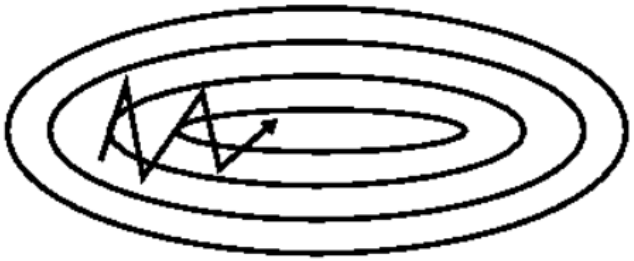
- learning rate $\eta$의 선택
  - learning rate를 너무 작거나 큰 고정된 값으로 선택할 경우, 각각 문제가 있다.
  - learning rate를 학습 과정에서 바꿀 수 있도록 미리 정해놓는 경우, 학습 데이터의 특성을 반영하지 못한다.
  - 모든 parameter들에 대해서 같은 learning rate가 반영되므로, 적절하지 못한 경우가 생긴다.
- Local Minima에 갇혀서 global minima로 수렴하지 못하는 경우

# Various Gradient Descent Optimizations

위 문제들을 해결하기 위하여, 아래의 다양한 optimizing strategy를 사용한다.

- Momentum
- Nesterov Accelerated Gradient
- Adagrad
- 등등등...

# Momentum

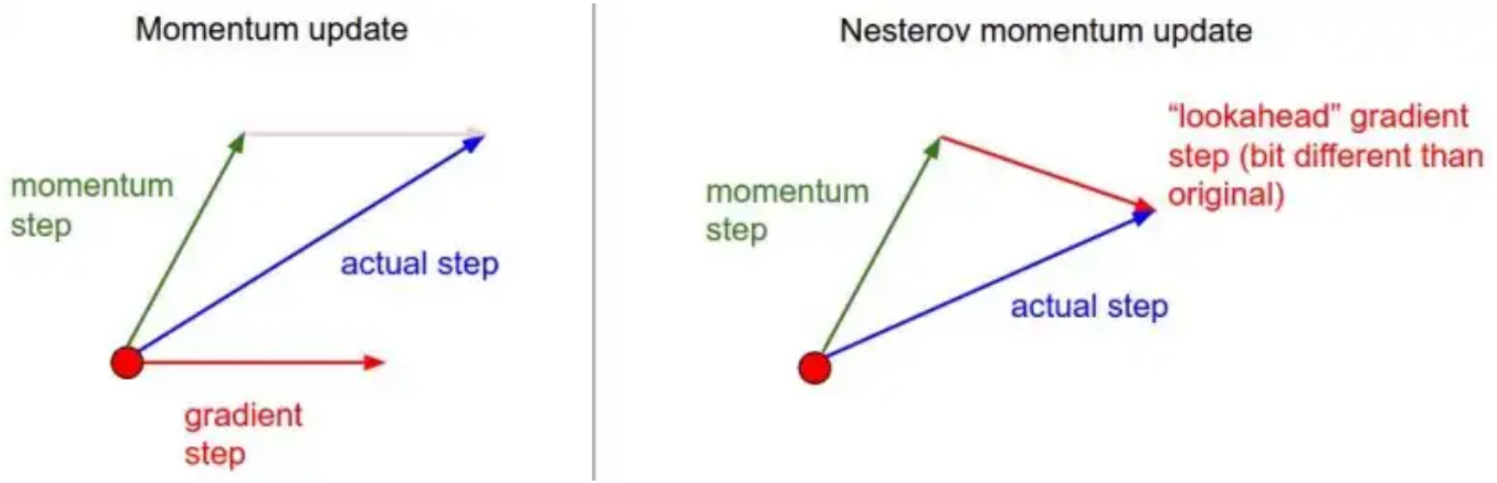


일반적으로, 위에서 제시된 SGD 방법들은 어떤 특정한 한 축에 대해서 곡면에 훨씬 steep할 경우, 학습이 느려지는 경향이 있다. 따라서 다음과 같이 기존의 움직임을 반영하는 일종의 *운동량* 을 추가적으로 더해준다.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J \tag{5}$$

$$\theta = \theta - v_t \tag{6}$$

# Nesterov Momentum




위 운동량을 개량한 버젼이 Nesterov Momentum이다. 이는 우리가 기존에 생각하던 momentum에 일종의 lookahead를 주는 것이다.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \quad (7)$$

$$\theta = \theta - v_t \quad (8)$$

# Adagrad

이때까지는 updating하는 방법에 대한 것이였다면, 이번에는 각 parameter에 따라서 다른 learning rate를 제공하자는 아이디어를 만든 것이다. 어떤 feature가 특정 learning rate에서 많이 바뀐다면, 그 feature에 대한 learning rate를 낮춤으로써 조금 더 안정적인 학습을 진행할 수 있으며, 반대로 거의 바뀌지 않는다면 조금 더 learning rate를 높여 더 빠른 학습을 꾀할 수 있다. 학습 과정은 다음과 같다.
각 파라미터마다 learning rate가 다르므로, 아래 식에서는 t를 학습 횟수, i를 parameter 인덱스로 사용하겠다.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \nabla J(\theta_{t,i}) \tag{9}$$

여기서, $G_{t,i}$는 i번째 파라미터가 t번 학습할 때까지 변한 값의 제곱의 합이며, $\epsilon$은 그저 division by zero 에러를 막기 위한 작은 값으로, 보통 $10^{-8}$ 정도의 값을 사용한다.

# Adadelta

Adagrad가 그 전의 모든 그래디언트의 변화를 다 저장했다면, Adadelta는 과거의 그래디언트 변화값을 덜 반영하고, 상대적으로 최근의 변화에 가중치를 주는 방법으로 학습을 진행한다. 즉,

$$h_t = \gamma h_{t-1} + (1-\gamma)\nabla J\theta_t \tag{10}$$

$$\delta\theta_{t,i} = \frac{-\eta}{h_{t,i-1}+\epsilon}\nabla J\theta_{t,i-1} \tag{11}$$

각 파라미터마다 learning rate가 다르므로, 아래 식에서는 t를 학습 횟수, i를 parameter 인덱스로 사용하겠다.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}+\epsilon}}\nabla J(\theta_{t,i}) \tag{12}$$

여기서, $G_{t,i}$는 i번째 파라미터가 t번 학습할 때까지 변한 값의 제곱의 합이며, $\epsilon$은 그저 division by zero 에러를 막기 위한 작은 값이다.